Robot Grasp Quality

August 24, 2020

1 Machine learning Capstone project

2 Project overview

The aim of the project is to predict the grasp quality of a robotic hand given the data captured during a series of experiments using Smart Grasping Sandbox (SGS). The data generated from the simulated experiment among many headers contains information about hand, finger, joint position and velocity which will be the primary key information I will be using to predict the next grasp.

3 Problem statement

How to predict the grasp of an object before perfoming an action Which mean how to position the robotic hand in space in such way that when the grasping of that particular object is necessary then it is done correctly.

3.1 Metrics

Good grasp occurs when the robot hand successfully grabs the red ball and does not drop it otherwise, it will be considered a bad grasp.

Total amount of time the robot performs successfully the action corresponds with the total number of experiments performed.

The ration of Good graps over the total number of experiments is shown in the picture below. Same applies for the bad grasp.

3.2 Analysis

3.2.1 Criteria

Criteria of analysis will be done based on the highest accuracy score. From the results yielded we can see that .... NN shows a % of XX compared to .... NN which shows a % of %

3.3 Benchmark

As a benchmark for my model I will use the highest % score that will be generated from my initial NN which will get compared with the results that will be yielded with another framework.

4 Explore the dataset

```
In [7]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [9]: # import the dataset
        csv_file = 'dataset/shadow_robot_dataset.csv'
        df = pd.read_csv(csv_file)

        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 992641 entries, 0 to 992640
Data columns (total 30 columns):
experiment_number      992641 non-null object
 robustness            992641 non-null float64
 H1_F1J2_pos           992641 non-null float64
 H1_F1J2_vel           992641 non-null float64
 H1_F1J2_eff           992641 non-null float64
 H1_F1J3_pos           992641 non-null float64
 H1_F1J3_vel           992641 non-null float64
 H1_F1J3_eff           992641 non-null float64
 H1_F1J1_pos           992641 non-null float64
 H1_F1J1_vel           992641 non-null float64
 H1_F1J1_eff           992641 non-null float64
 H1_F3J1_pos           992641 non-null float64
 H1_F3J1_vel           992641 non-null float64
 H1_F3J1_eff           992641 non-null float64
 H1_F3J2_pos           992641 non-null float64
 H1_F3J2_vel           992641 non-null float64
 H1_F3J2_eff           992641 non-null float64
 H1_F3J3_pos           992641 non-null float64
 H1_F3J3_vel           992641 non-null float64
 H1_F3J3_eff           992641 non-null float64
 H1_F2J1_pos           992641 non-null float64
 H1_F2J1_vel           992641 non-null float64
 H1_F2J1_eff           992641 non-null float64
 H1_F2J3_pos           992641 non-null float64
 H1_F2J3_vel           992641 non-null float64
 H1_F2J3_eff           992641 non-null float64
 H1_F2J2_pos           992641 non-null float64
 H1_F2J2_vel           992641 non-null float64
 H1_F2J2_eff           992641 non-null float64
 measurement_number    992641 non-null int64
dtypes: float64(28), int64(1), object(1)
memory usage: 227.2+ MB
```

In [10]: `df.head(3)`

Out[10]:

```
                     experiment_number  robustness  H1_F1J2_pos  \
0  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.118209
1  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.152945
2  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.162168

   H1_F1J2_vel  H1_F1J2_eff  H1_F1J3_pos  H1_F1J3_vel  H1_F1J3_eff  \
0     6.838743     1.454113     0.302276   -18.738705          0.0
1     5.997176     1.098305     0.308893   -14.173090          0.0
2     5.302321     0.999142     0.314331   -13.093510          0.0

   H1_F1J1_pos  H1_F1J1_vel         ...          H1_F2J1_pos  \
0    -0.032352     0.127232         ...             0.109246
1    -0.027381     0.273711         ...             0.105656
2    -0.025808     0.184343         ...             0.103620

   H1_F2J1_vel  H1_F2J1_eff  H1_F2J3_pos  H1_F2J3_vel  H1_F2J3_eff  \
0     0.042166     0.041517     0.439459   -13.975613          0.0
1    -0.130178     0.075700     0.446421   -17.618561          0.0
2    -0.162815     0.095730     0.439690   -13.031110          0.0

   H1_F2J2_pos  H1_F2J2_vel  H1_F2J2_eff  measurement_number
0     0.177114     5.456443     1.493776                   0
1     0.176817     5.130892     1.493497                   1
2     0.174343     5.650662     1.523433                   2

[3 rows x 30 columns]
```

In [11]: `df.tail(3)`

Out[11]:

```
                          experiment_number  robustness  H1_F1J2_pos  \
992638  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.130326
992639  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.149129
992640  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.109327

        H1_F1J2_vel  H1_F1J2_eff  H1_F1J3_pos  H1_F1J3_vel  \
992638     7.749944     1.850211     0.251145   -21.228001
992639     6.136092     1.646002     0.259715   -17.988816
992640    -7.797566     0.000000     0.256289    19.186920

        H1_F1J3_eff  H1_F1J1_pos  H1_F1J1_vel         ...           \
992638     0.000000    -0.027532    -0.020924         ...
992639     0.000000    -0.024531     0.080302         ...
992640     0.761764    -0.021199     0.187938         ...

        H1_F2J1_pos  H1_F2J1_vel  H1_F2J1_eff  H1_F2J3_pos  \
992638    -0.064897     0.035043     0.011649     0.371388
```

```
992639      -0.067001      0.041980      0.032763      0.371682
992640      -0.063383      0.079873     -0.003048      0.379547

         H1_F2J3_vel   H1_F2J3_eff   H1_F2J2_pos   H1_F2J2_vel  \
992638      -5.145467      0.000000      0.205514      2.087309
992639      -5.562895      0.000000      0.205412      2.323993
992640       5.288929      0.129144      0.221894     -1.628677

         H1_F2J2_eff   measurement_number
992638      0.458245                   27
992639      0.461640                   28
992640      0.000000                   29

[3 rows x 30 columns]
```

In [5]:
```
csv_file = 'dataset/shadow_robot_dataset.csv'
# either use header = 0 or dont use any header argument.
df = pd.read_csv(csv_file, header = 0)
```

In [2]:
```
# header = 1 means consider second line of the dataset as header.
df = pd.read_csv(csv_file, header = 1)
```

Out[2]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe117f514e0>
```

5 Explore dataset

5.0.1 Class distributions

In [31]:
```
# locating important parameters iloc[column, rows]
training = df.iloc[1:10]
training.head()
```

Out[31]:
```
                  experiment_number  robustness  H1_F1J2_pos  \
1  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.152945
2  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.162168
3  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.137684
4  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.161747
5  2ccc5f2c534f4be2b329eada685ce311   85.758903     0.142037

   H1_F1J2_vel  H1_F1J2_eff  H1_F1J3_pos  H1_F1J3_vel  H1_F1J3_eff  \
1     5.997176     1.098305     0.308893   -14.173090          0.0
2     5.302321     0.999142     0.314331   -13.093510          0.0
3     6.504519     1.256002     0.304333   -16.948796          0.0
4     4.899113     0.999313     0.315815   -13.700695          0.0
5     6.244418     1.209869     0.306419   -16.266108          0.0

   H1_F1J1_pos  H1_F1J1_vel      ...       H1_F2J1_pos  \
1    -0.027381     0.273711      ...          0.105656
```

```
2    -0.025808    0.184343   ...          0.103620
3    -0.027398    0.121100   ...          0.106332
4    -0.025698    0.079876   ...          0.104104
5    -0.027343    0.144840   ...          0.105687

   H1_F2J1_vel  H1_F2J1_eff  H1_F2J3_pos  H1_F2J3_vel  H1_F2J3_eff  \
1    -0.130178     0.075700     0.446421   -17.618561          0.0
2    -0.162815     0.095730     0.439690   -13.031110          0.0
3    -0.186364     0.068382     0.445833   -11.763374          0.0
4    -0.216307     0.090358     0.438578   -15.347191          0.0
5    -0.166225     0.075026     0.442759   -13.477787          0.0

   H1_F2J2_pos  H1_F2J2_vel  H1_F2J2_eff  measurement_number
1     0.176817     5.130892     1.493497                   1
2     0.174343     5.650662     1.523433                   2
3     0.180723     5.267410     1.455800                   3
4     0.164628     6.339569     1.627478                   4
5     0.176201     5.781911     1.506166                   5

[5 rows x 30 columns]
```

In [ ]:

In [6]:
```
df.tail(10)
```

Out[6]:
```
                         experiment_number  robustness  H1_F1J2_pos  \
992631  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.125458
992632  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.127350
992633  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.132881
992634  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.131513
992635  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.129801
992636  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.130388
992637  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.128054
992638  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.130326
992639  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.149129
992640  2ce8d77087094b11a8596c53f1c2df15   96.585662     0.109327

        H1_F1J2_vel  H1_F1J2_eff  H1_F1J3_pos  H1_F1J3_vel  \
992631     7.590437     1.897293     0.251336   -20.584301
992632     7.135461     1.873800     0.254291   -18.860431
992633     6.857425     1.815702     0.256093   -19.040201
992634     6.966961     1.830468     0.252527   -19.167209
992635     7.335149     1.851264     0.252165   -20.034763
992636     7.184142     1.843887     0.253100   -19.537161
992637     7.520762     1.870689     0.251921   -20.113798
992638     7.749944     1.850211     0.251145   -21.228001
992639     6.136092     1.646002     0.259715   -17.988816
992640    -7.797566     0.000000     0.256289    19.186920
```

	H1_F1J3_eff	H1_F1J1_pos	H1_F1J1_vel	...	\
992631	0.000000	-0.028187	0.016215	...	
992632	0.000000	-0.027845	0.031700	...	
992633	0.000000	-0.027372	0.027302	...	
992634	0.000000	-0.026947	-0.019985	...	
992635	0.000000	-0.028068	0.044246	...	
992636	0.000000	-0.027824	0.050854	...	
992637	0.000000	-0.028146	0.034234	...	
992638	0.000000	-0.027532	-0.020924	...	
992639	0.000000	-0.024531	0.080302	...	
992640	0.761764	-0.021199	0.187938	...	

	H1_F2J1_pos	H1_F2J1_vel	H1_F2J1_eff	H1_F2J3_pos	\
992631	-0.063555	-0.017697	-0.002304	0.372126	
992632	-0.064275	-0.031236	0.004764	0.371796	
992633	-0.065521	0.034403	0.017878	0.373136	
992634	-0.064968	0.077494	0.012780	0.372523	
992635	-0.064540	-0.019866	0.007529	0.372625	
992636	-0.064553	-0.044033	0.007415	0.373140	
992637	-0.064192	-0.006108	0.004178	0.372934	
992638	-0.064897	0.035043	0.011649	0.371388	
992639	-0.067001	0.041980	0.032763	0.371682	
992640	-0.063383	0.079873	-0.003048	0.379547	

	H1_F2J3_vel	H1_F2J3_eff	H1_F2J2_pos	H1_F2J2_vel	\
992631	-4.642603	0.000000	0.208041	1.889879	
992632	-5.252718	0.000000	0.207601	2.209818	
992633	-5.203855	0.000000	0.206951	2.157384	
992634	-5.010247	0.000000	0.205460	2.024550	
992635	-4.842305	0.000000	0.208904	2.008012	
992636	-4.915019	0.000000	0.209603	2.067703	
992637	-4.841139	0.000000	0.207827	1.989526	
992638	-5.145467	0.000000	0.205514	2.087309	
992639	-5.562895	0.000000	0.205412	2.323993	
992640	5.288929	0.129144	0.221894	-1.628677	

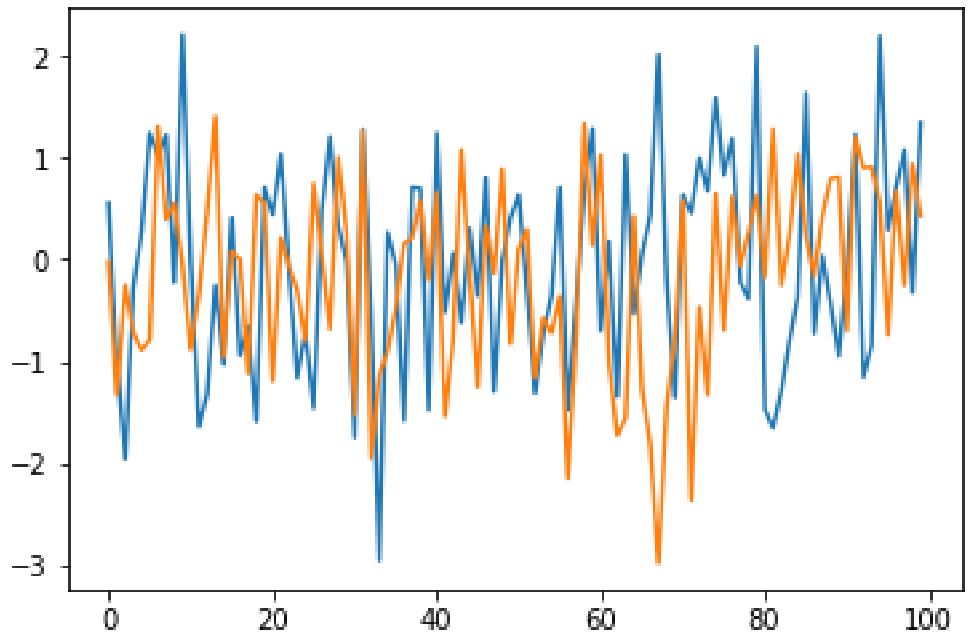
	H1_F2J2_eff	measurement_number
992631	0.431011	20
992632	0.438603	21
992633	0.444583	22
992634	0.458159	23
992635	0.423553	24
992636	0.417153	25
992637	0.434158	26
992638	0.458245	27
992639	0.461640	28
992640	0.000000	29

```
[10 rows x 30 columns]
```

6 Exploratory visualization

Plot the data from the dataset

```
In [27]: import pandas
         import matplotlib.pyplot as plt
         # dataset = pandas.read_csv('dataset/shadow_robot_dataset.csv', usecols=[1], engine='py
         plt.plot(df)
         plt.show()
```



6.1 Algorithms and Techniques

To implement the ... NN I will use the common technique of splitting the test and train set...
X_test, X_train, Y_test, Y_train
the above will get fed to the NN.

7 Keras & Sklearn

```
In [25]: from keras.models import Sequential
         from keras.layers import Dense
         from keras.callbacks import TensorBoard
         from keras.layers import *
```

```python
import numpy

from sklearn.model_selection import train_test_split

# Ignore the first row and column
dataset = numpy.loadtxt("dataset/shadow_robot_dataset.csv", skiprows = 1, usecols = ran
```

Since my output vector expected is the grasp robustness. I will read the header of my CSV file and then collect and store those values into a list. The list will get converted to numpy array which will serve for the output vector containing the predicted grasp robustness.

```python
In [26]: # csv_file = 'dataset/shadow_robot_dataset.csv'
         # df = pd.read_csv(csv_file)

         # Getting the header

         with open('dataset/shadow_robot_dataset.csv', 'r') as f:
             header = f.readline()
         # remove whitespace characters
         header = header.strip("\n").split(',')
         header
         header = [i.strip(" ") for i in header]

         # use velocity and effort
         saved_cols = []
         for index, col in enumerate(header[1:]):
             if ("vel" in col) or ("eff" in col):
                 saved_cols.append(index)

         new_X = []
         for x in dataset:
             new_X.append([x[i] for i in saved_cols])

         # X - split of the dataset
         X = numpy.array(new_X)
```

```python
In [1]: import pandas as pd
        csv_file = 'dataset/shadow_robot_dataset.csv'
        df = pd.read_csv(csv_file)
```

```python
In [9]: # Now let's split the test and train set
        Y = dataset [:, 0]
```

```python
In [10]: # Provide a random seed
         rnd_seed = 6

         # dataset split
         X_test, X_train, Y_test, Y_train = train_test_split(X, Y, test_size = 0.30, random_stat
```

```
# Good grasp threshold for stability

good_grasp = 50

# Store good and best grasp results
itemindex = numpy.where(Y_test > 1.05 * good_grasp)

best_grasps = X_test[itemindex[0]]

itemindex = numpy.where(Y_test <= 0.95 * good_grasp)

bad_grasps = X_test[itemindex[0]]


# splitting the grasp quality for stable or unstable grasps
Y_train = numpy.array([int(i > good_grasp) for i in Y_train])
Y_train = numpy.reshape(Y_train, (Y_train.shape[0],))

Y_test = numpy.array([int(i > good_grasp) for i in Y_test])
Y_test = numpy.reshape(Y_test, (Y_test.shape[0],))
```

8 Building the model

```
In [11]: # building the model
         model = Sequential()

         model.add(Dense(20*len(X[0]), use_bias = True, input_dim = len(X[0]), activation = 'rel
         model.add(Dropout(0.5))

         model.add(Dense(1, activation = 'sigmoid'))

         # Compiling the model
         model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

9 Training the model

```
In [24]: model.fit(X_train, Y_train, validation_split = 0.20, epochs = 50,
                   batch_size = 500000)
```

```
Train on 238234 samples, validate on 59559 samples
Epoch 1/50
238234/238234 [==============================] - 2s 10us/step - loss: 0.8887 - acc: 0.5071 - val
Epoch 2/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.7531 - acc: 0.4583 - val_
Epoch 3/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.6271 - acc: 0.5157 - val_
Epoch 4/50
```

```
238234/238234 [==============================] - 0s 1us/step - loss: 0.5336 - acc: 0.5618 - val_
Epoch 5/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.4704 - acc: 0.5933 - val_
Epoch 6/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.4284 - acc: 0.6156 - val_
Epoch 7/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.4024 - acc: 0.6692 - val_
Epoch 8/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3879 - acc: 0.7672 - val_
Epoch 9/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3810 - acc: 0.8316 - val_
Epoch 10/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3777 - acc: 0.9041 - val_
Epoch 11/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3749 - acc: 0.9390 - val_
Epoch 12/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3743 - acc: 0.9445 - val_
Epoch 13/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3753 - acc: 0.9454 - val_
Epoch 14/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3753 - acc: 0.9463 - val_
Epoch 15/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3760 - acc: 0.9468 - val_
Epoch 16/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3762 - acc: 0.9473 - val_
Epoch 17/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3760 - acc: 0.9479 - val_
Epoch 18/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3743 - acc: 0.9486 - val_
Epoch 19/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3729 - acc: 0.9492 - val_
Epoch 20/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3739 - acc: 0.9499 - val_
Epoch 21/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3724 - acc: 0.9509 - val_
Epoch 22/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3716 - acc: 0.9516 - val_
Epoch 23/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3684 - acc: 0.9524 - val_
Epoch 24/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3666 - acc: 0.9536 - val_
Epoch 25/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3637 - acc: 0.9547 - val_
Epoch 26/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3629 - acc: 0.9557 - val_
Epoch 27/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3592 - acc: 0.9567 - val_
Epoch 28/50
```

```
238234/238234 [==============================] - 0s 1us/step - loss: 0.3568 - acc: 0.9574 - val_
Epoch 29/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3549 - acc: 0.9583 - val_
Epoch 30/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3531 - acc: 0.9590 - val_
Epoch 31/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3501 - acc: 0.9596 - val_
Epoch 32/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3472 - acc: 0.9600 - val_
Epoch 33/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3454 - acc: 0.9602 - val_
Epoch 34/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3429 - acc: 0.9608 - val_
Epoch 35/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3402 - acc: 0.9610 - val_
Epoch 36/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3380 - acc: 0.9614 - val_
Epoch 37/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3354 - acc: 0.9618 - val_
Epoch 38/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3333 - acc: 0.9623 - val_
Epoch 39/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3317 - acc: 0.9628 - val_
Epoch 40/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3297 - acc: 0.9631 - val_
Epoch 41/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3284 - acc: 0.9631 - val_
Epoch 42/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3257 - acc: 0.9636 - val_
Epoch 43/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3255 - acc: 0.9636 - val_
Epoch 44/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3242 - acc: 0.9642 - val_
Epoch 45/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3215 - acc: 0.9641 - val_
Epoch 46/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3199 - acc: 0.9643 - val_
Epoch 47/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3184 - acc: 0.9647 - val_
Epoch 48/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3172 - acc: 0.9648 - val_
Epoch 49/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3147 - acc: 0.9652 - val_
Epoch 50/50
238234/238234 [==============================] - 0s 1us/step - loss: 0.3136 - acc: 0.9651 - val_
```

```
Out[24]: <keras.callbacks.History at 0x7f5fc3078748>
```

In [25]:
```python
# I will save the trained model trained with Keral library for later use
import h5py
model.save("./keras_model.h5")
```

In [27]:
```python
# evaluating the model
score = model.evaluate(X_test, Y_test)

print("%s : %.3f%%" % (model.metrics_names[1], score[1]*100))
```

```
694848/694848 [==============================] - 35s 50us/step
acc : 97.137%
```

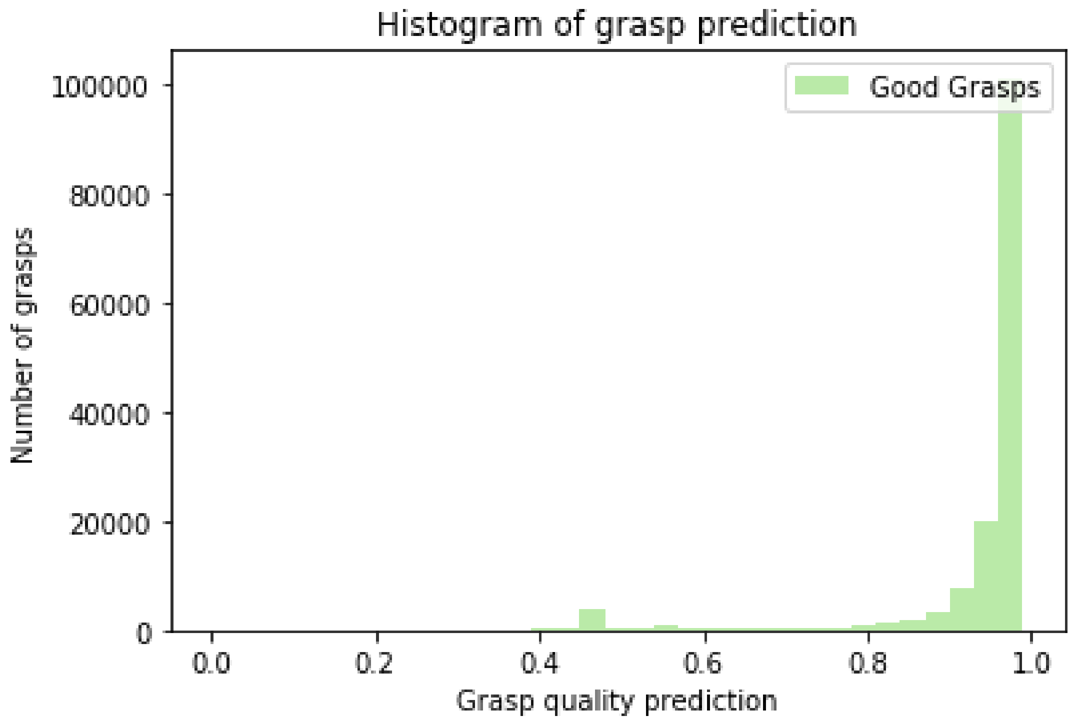
In [29]:
```python
# plotting predictions
predictions = model.predict(best_grasps)

%matplotlib inline
import matplotlib.pyplot as plt

plt.hist(predictions,
         color='#77D653',
         alpha=0.5,
         label='Good Grasps',
         bins=np.arange(0.0, 1.0, 0.03))

plt.title('Histogram of grasp prediction')
plt.ylabel('Number of grasps')
plt.xlabel('Grasp quality prediction')
plt.legend(loc='upper right')

plt.show()
```



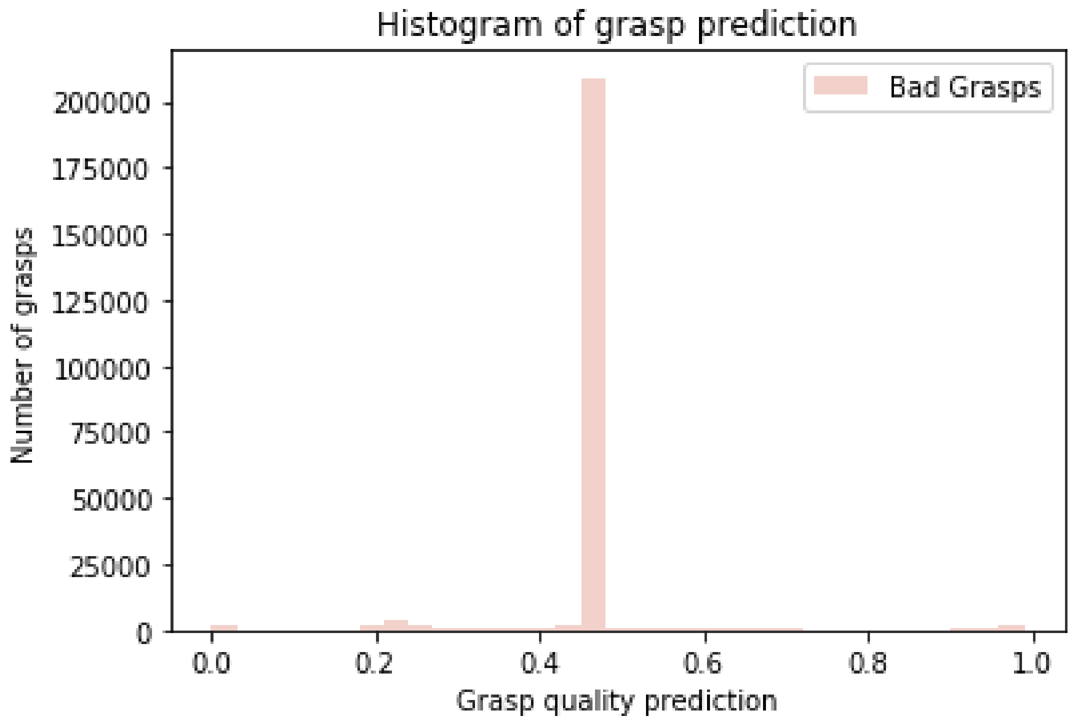
From the graph above it can bee seen that most of the grasps are correctly predicted as stable <0.8

In [30]:
```
# What about plotting the unstable grasps?
# Unstable grasps
predictions_bad_grasp = model.predict(bad_grasps)


# Plot a histogram of defender size
plt.hist(predictions_bad_grasp,
         color = '#D66751',
         alpha = 0.3,
         label = 'Bad Grasps',
         bins = np.arange(0.0, 1.0, 0.03))

plt.title('Histogram of grasp prediction')
plt.ylabel('Number of grasps')
plt.xlabel('Grasp quality prediction')
plt.legend(loc='upper right')

plt.show()
```



9.1 Results

9.1.1 Model evaluation and validation

After evaluating the model the prediction accuracy is fairly high 0.97 thus the number of good and bad grasps is close to the confident values obtained from the dataset.

9.1.2 References

[1] Carlos Rubert, Daniel Kappler, Jeannette Bohg and Antonio Morales: Grasp success prediction

[2] Jialiang (Alan) Zhao, Jacky Liang, and Oliver Kroemer: Towards Precise Robotic Grasping by P

[3] Alex Keith Goins: thesis work

[4] https://www.kaggle.com/ugocupcic/grasp-quality-prediction/

[5]

[6] Shadow robot Building a sandbox for hand-robot training

[7] Google developers Common ML problems